AD-A008 153

A COMMAND LANGUAGE PROCESSOR FOR FLEXIBLE INTERFACE DESIGN

Russell J. Abbott

University of Southern California

Prepared for:

Advanced Research Projects Agency

September 1974

DISTRIBUTED BY:

NTIS

National Technical Information Service
U. S. DEPARTMENT OF COMMERCE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER ISI/RR-74-24	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER AD/A008153
4. TITLE (and Subtitle) A COMMAND LANGUAGE PROCESSOR FOR FLEXIBLE INTERFACE DESIGN		5. TYPE OF REPORT & PERIOD COVERED Research
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Russell J. Abbott		8. CONTRACT OR GRANT NUMBER(s) DAHC 15 72 C 0308
9. PERFORMING ORGANIZATION NAME AND ADDRESS USC/Information Sciences Institute 4676 Admiralty Way Marina Del Rey, CA 90291		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order #2223/1
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, Virginia 22209		12. REPORT DATE September 1974
		13. NUMBER OF PAGES 68
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) ---------------		15. SECURITY CLASS. (of this report) ---------------
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE ---------------

16. DISTRIBUTION STATEMENT (of this Report)

This document approved for public release and sale; distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

----

18. SUPPLEMENTARY NOTES

----

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Command language, command language processor, dialogue grammar, interactive grammar, human-engineered, flexible user interface, user interface, interactive system.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(OVER)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

A functional specification for a computer-based, interactive Command Language Processor (CLP) is presented. The CLP is the language processing component of a user-oriented interface (called the Agent) to a sophisticated, text preparation and message processing service. The CLP differs from most language processors in that it operates with a dialogue grammar and takes the entire interaction between user and Agent as its source parse string. The parse is ongoing and proceeds along with the interaction. Within this framework, the CLP provides: pronominal referencing, situation dependent prompts and, on request, a contextual review of the entire interaction.

As the user and Agent interact, the CLP develops a lexicon of the user's name space. Using that lexicon, the CLP recognizes abbreviated and incompletely spelled words.

A facility is provided to define macro commands presented in example form by the user.

ARPA ORDER NO. 2223

ISI/RR-74-24
September 1974

Russell J. Abbott

A Command Language Processor for Flexible Interface Design

INFORMATION SCIENCES INSTITUTE

4676 Admiralty Way/Marina del Rey/California 90291
(213) 822-1511

UNIVERSITY OF SOUTHERN CALIFORNIA

THIS RESEARCH IS SUPPORTED BY THE ADVANCED RESEARCH PROJECTS AGENCY UNDER CONTRACT NO. DAHC15 72 C 0308, ARPA ORDER NO. 2223, PROGRAM CODE NO. 3D30 AND 3P10.

CONTENTS

CONTENTS

CONTENTS

FIGURES

PREFACE

The Information Automation project at USC/Information Sciences Institute is currently developing methods to automate various information handling tasks, with particular emphasis on message processing for military command, control, and communications [1,2]. The project, sponsored by ARPA, is an integral part of the client's and ISI's overall program to explore the use of computer technology and methodology in military environments.

This study is one of a planned collection of reports that describes the current status and plans of the Information Automation project. This report describes the major interface components of the military message processing service. In particular three modules are discussed: the Command Language Processor, the Input Interface and the Screen Control Module.

For a brief overview, one may read the Summary alone. For a fuller picture, one should read Part 1: Introduction and Part 2: The Service Components and How They Are Connected. For a good picture of the services provided by the Command Language Processor, Part 4 is recommended. Readers interested in more detailed functional specifications of the three subject modules are referred to Part 3.

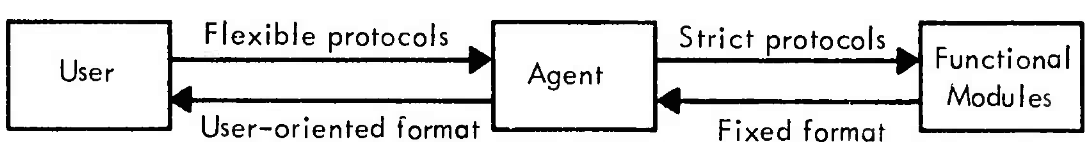
The Information Automation project includes an investigation into the effect of various command languages on user populations. For that reason the military message service is designed with a number of different command languages. Part 5 of this report describes how those command languages are specified.

Most of the work reported in this study and its accompanying documents reflects an orientation which applies to other services besides the message service. Part 5 of this document also discusses how an arbitrary service may be connected to the service framework developed.

SUMMARY

The Information Automation Project's military message processing service is a prototype computer service. Its goal is to provide computer-naive users with simple, direct and intelligible access to computing power. One may view an interactive computer service as consisting of two major components: the user and the service's Functional Modules [3]. The goal of the message service is to simplify the user's access to the Functional Modules. To achieve this goal the service is designed with a single, two-way, special services interface which represents the user to the Functional Modules and the Functional Modules to the user.

The interface is called the Agent. It consists of a number of components: The Command Language Processor (CLP), the Input Interface to the CLP, the output interface or the Screen Control Module, the User Monitor [4] and the Tutor [5]. Each of these components contributes to making the user and the service Functional Modules mutually intelligible. In effect, the Agent is a soft and forgiving buffer between the precise formality required by the Functional Modules and the flexible imprecision needed by uncertain human users. (See figure below.)



The Agent provides a flexible interface between the user and the service functional modules. It interprets each to the other.

This document provides functional descriptions for the CLP, the Input Interface and the Screen Control Module. The latter two are described briefly. The main emphasis is on the CLP and its interfaces. Central to the translation between the user's informality and the formal requirements of the Functional Modules, the CLP has as its main job the development and maintenance of a formal analysis of the user's interactions.

SUMMARY

The following paragraphs summarize the main contents of this report.

Part 1 presents an overview of the techniques, services and facilities embodied by the Agent. The basic mechanism may be summarized by noting that the entire message service executes in a syntax-directed manner through a single interpreter (the CLP). The CLP analyzes and executes a single dynamically modified string resulting from and reflecting the dialogue between the user and the Agent.

Part 2 provides a brief overview of the various service components and interfaces. It takes the reader on a guided tour of the information flow paths through the entire service. There are three major components and two interfaces. The components are the user, the Agent and the Functional Modules; the interfaces are those between the user and Agent and between the Agent and Functional Modules.

The Agent consists of the CLP, the Tutor and the User Monitor. The CLP is the single language processor in the service. All other modules use its language analyses. The Tutor is the central and sole source of tutorial assistance in the service. If the user needs operational information, the Tutor supplies it. The User Monitor supervises the user and suggests additional ways for the Tutor to be helpful.

The Input Interface and the Screen Control Module provide, respectively, the input and output interfaces linking the user and Agent. The Input Interface provides a uniform and omnipresent input and editing medium for the user, while the Screen Control Module performs a similar function for the Agent (and hence the entire service).

This part also traces the various possible information flow paths through these modules and indicates how each component and interface makes its contribution.

Part 3 provides more detail on the three modules which are the primary subject of this report. The Input Interface is a simple but complete editor. It includes commands to move the cursor, to insert, delete and replace text. Its editing facilities are defined to be straightforward, intuitive and valuable, especially for the small-scale editing of text.

The Screen Control Module provides facilities for service components to communicate with the user. It permits the definition of multiple viewing windows. Each window is described by a number of window components programmable for the benefit of the service Functional Modules. Yet they are display-oriented and do not require device-like coding. At this point, the specifications of the Input Interface and the Screen Control Modules are concluded.

SUMMARY

The CLP is a general, rule-driven interpreter. Unlike most language processors, however, it operates on a dynamic input string and an ever-changing parse tree rather than on a static string and fixed parse tree. The CLP operates with a dialogue grammar which reflects the entire dialogue between the user and Agent. The CLP is always in the state of advancing its parse of the expanding user/Agent interaction.

Part 4 presents details of the control structures which govern a terminal session. The rules of the dialogue grammar are divided into groups called contexts which perform two main functions. First, they disambiguate dialogue fragments; second, they provide a structure to the dialogue.

In this part are described the various services CLP offers the user:

- The CLP provides situation-dependent PROMPTs when the user requests them.
- The user may request the CLP to ABORT one or more of his activities.
- Likewise, the user may request that the CLP UNDO some action.
- If asked, the CLP will REVIEW the session with the user and respond to WHERE-AM-I with a comforting blanket of context.
- The CLP permits EXECUTION FROM A FILE, including command programs with parameters accepted from the terminal.
- The CLP maintains a LEXICON with which it provides facilities for automatic ABBREVIATION, argument and command RECOGNITION, and display of the USER NAME SPACE and subSPACES.
- In conjunction with the Tutor and User Monitor, the CLP distinguishes between commands known to the user and other commands.
- The CLP permits the definition of MACROs. From an example of its use, the CLP generates the intended macro. Computer-naive users should have no trouble with this heretofor programmer-like operation.
- The user is permitted to make PRONOMINAL REFERENCES. The CLP finds the antecedent.
- Facilities are provided for the user to continue work from one terminal session to another as if there had been no interruption. Whether or not the user takes advantage of that particular device, the more the user interacts with the CLP, the better service he will receive.

Part 5 discusses two interfaces: one between the Functional Modules and the Agent; the other, the methodology for defining the command languages.

A Functional Module is defined to the CLP in terms of functions grouped into contexts. Each function is defined via a function descriptor. Each context has a context descriptor.

From these function descriptors, and taking into account the particular needs of the user audience, a language designer creates commands. A command may be in any one of four command forms: functional positional notation, functional keyword notation, simple English or function key form. The language designer chooses a language form to create commands.

In the creation of commands, the language designer specifies the command vocabulary and the argument characteristics. For each argument the designer has a choice of various default and visibility options. Through the careful use of these options the language designer tailors the commands to a particular user group.

The language designer also has the option of making the entire interaction either CLP-driven or user-driven. If CLP-driven, the CLP prompts the user with command frames in a fill-in-the-blanks format.

The language designer is free to map the commands and the Functional Module functions together on a one-to-one, one-to-many or many-to-one basis.

In conclusion, the primary aim of the message service is to provide computation and data processing services to users who have little knowledge of computers and data processing. This document describes some of the major components of the message processing service and shows how the CLP, the Input Interface and the Screen Control Module help achieve this important goal.

1. INTRODUCTION

The Information Automation project's military message processing service is a prototype of a class of computer services in which computer-naive users are provided with simple, direct and intelligible access to computing power. Different services in this class provide the user with varying sorts of computational capabilities. The military message service takes message processing as its demonstration vehicle. One may view an interactive computer service as consisting of two major components: the user and the service's Functional Modules [3]. A goal of the military message service is to simplify the user's access to Functional Modules. To achieve this goal the service is designed with a single, two-way, special services interface, which represents the user to the Functional Modules and the Functional Modules to the user.

The interface is called the Agent. It consists of a number of components: The Command Language Processor (CLP), the Input interface to the CLP, the output interface or the Screen Control Module , the User Monitor [4] and the Tutor [5]. Each of these components contributes to making the user and the service Functional Modules mutually intelligible. In effect, the Agent is a soft and forgiving buffer between the precise formality required by the Functional Modules and the flexible imprecision needed by uncertain human users. (See Fig. 1)

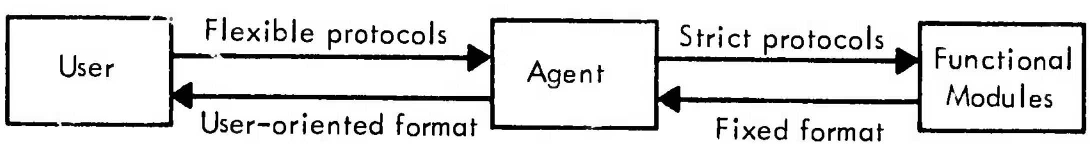


Figure 1. The Agent provides a flexible interface between the user and the service functional modules. It interprets each to the other.

This document provides functional descriptions for the CLP, the Input Interface and the Screen Control Module. The latter two are described briefly. The main emphasis is on the CLP and its interactions with the other components of the service. Central to the translation between the user's informality and the formal requirements of the Functional Modules, the CLP has as its main job the development and maintenance of a formal analysis of the user's interactions. Here we provide a brief overview of the CLP's approach.

OVERVIEW OF THE COMMAND LANGUAGE PROCESSOR (CLP)

The CLP is in fact a language processor. It operates with a grammar and provides a formal parse of the user's input characters. It differs from most computer language processors in that the input string that it parses is not fixed. The CLP parses the entire interaction between the user and agent as a single entity.

The Dialogue Grammar

The CLP produces its analysis through the use of a dialogue grammar. A dialogue grammar provides a formal framework for expressing the syntactic and semantic structure of an interactive dialogue in the same way that a standard grammar provides a formal framework for expressing the syntactic and semantic structure of a computer language. The CLP's dialogue grammar has as its domain an entire terminal session. (In some cases--see Part 4, *Session-to-session Continuity*--more than one terminal session is included in a single parse tree.) Thus the complete interaction between the user and Agent is included as part of the context for each successive analysis of the user's input.

The Session String

The CLP uses the dialogue grammar to parse the user's input. However, unlike the parse of a program by a compiler, which can complete its parse before deciding on the well-formedness of the input, the CLP must interact with the user during the parse. As the interaction progresses, the parse proceeds further. The user and the CLP collaborate to build a partial parse tree of their interaction to date as analyzed by the dialogue grammar. One path through this parse tree from left to right is called the session string.

The session string represents the highest level of the parse so far. That is, at the far left of the session string there is a single element representing the entire log-on procedure. Other elements represent other completed contexts of interaction. Toward the right-hand side of the session string, elements represent more detailed components of the interaction. One finds individual commands which the user has given in his current interaction context. There are subcommands shown if the user is currently working on the subcommands of a given command. Even individual arguments appear in the session string if the CLP and the user are currently working together on the specification of a command.

The session string represents the CLP's record of the user's view of the session. That is, the user is most concerned with the specific details of his current and recent activities. He remembers, probably, only more generally, his activities at the start of the session. The details of those activities are available elsewhere in the parse tree but they are not in the session string.

Whenever the user types new input, that input is concatenated to the session string on the right. As the parse progresses and the input is analyzed, the session string rises in the parse tree.

The session string often serves for the CLP as a scratch pad for interacting with the user. The real scratch-pad-like facility is the terminal screen. The CLP and the user communicate with each other through it. But the CLP uses the session string as its record of those communications. As far as the CLP is concerned, the action really occurs in the session string, and the terminal screen is used as a way of communicating that action to the user. Both the user and the CLP modify the session string. The user adds to the session string whenever he types additional characters. The CLP changes the session string in two ways. First, each time a dialogue grammar rule applies successfully, the session string changes to reflect that rule application. Second, whenever the Agent produces an interactive response to the user, that response is recorded by one or more elements added to the session string. Thus the session string truly reflects the interaction between the user and the CLP (see Fig. 2).

Summary

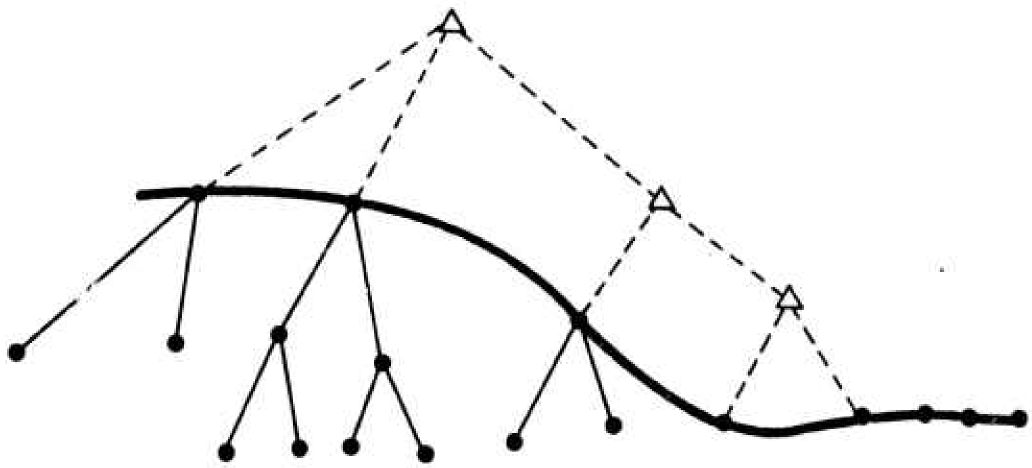
In language processing, a parse tree represents an analysis of some language sample. Unfortunately, that analysis is static; a language processor works on an input string and produces (or fails to produce) a parse. Even interactive processors eventually fail or succeed to parse the input. The dialogue grammar and session string method approaches the problem of language analysis from a new direction. The analysis is not only interactive but also dynamic, ongoing rather than a single event. This ongoingness relieves both the user and the CLP of many burdens and simultaneously provides many attractive advantages. Some of these are discussed below and throughout the remainder of this report.

SERVICE-WIDE CONSISTENCIES

The Agent framework, session string and dialogue grammar permit the military message service to provide the single, consistent, user-oriented interface which is its goal. The entire service operates in a syntax-driven manner from the session string. So all the Functional Modules have the same general feel to the user. This methodology allows a number of other service-wide consistencies.

The Input Interface

At his terminal, the user always has available to him a simple, small, but complete set of editing commands supplied by the Input interface. No matter what the user is typing or when he is typing it, these same editing facilities are available to him.



Δ Anticipated syntactic units

● Analyzed syntactic units

– – – Parse tree anticipated structure

——— Parse tree completed structure

▬▬ Session string

Figure 2. A partial parse tree and session string

Help, Undo, Abort and Tutor

The message service provides a number of ways for the user to ask for assistance or to modify his current activities. The Tutor, through the Help command, provides assistance; and the CLP's Abort and Undo commands permit the user to modify his current actions. These too are always available to him. Since they operate with respect to the session string, they are provided to the user at the level of specificity which he wants. The consistent flavor and ubiquity of these services are a comfort, especially to the naive user.

Context Mechanism

A context mechanism unites the dialogue grammar with the structure of the Functional Modules. It defines a control structure for the interaction in much the same way that sequentiality, hierarchies and subroutines do for programming languages. However, the context structure is not rigid as are subroutines. The CLP is able to move up and down and back and forth within the user's context hierarchy as the user jumps, with relative freedom, from context to context. This structure, with freedom, is also consistently available to the user.

Pronouns

Because the session string records commands, argument data types and contexts, the CLP is able to find the antecedents of pronouns. The user may make pronominal references both explicitly and by default.

LANGUAGE FEATURES

The CLP's linguistic framework is designed to provide four other important features.

Multiple Language Forms for Interaction

One goal of the message service is to provide language form variation to increase user efficiency and effectiveness. The User Monitor and the Tutor are mainly responsible for discovering needed improvements. They collaborate to suggest language form changes to the user. The CLP's structure permits the introduction of a number of different interaction language forms without a major reorganization.

Macros

The CLP operates in a linguistic universe in which lexical items are identified by data type. Taking full advantage of this framework, the CLP allows the user great freedom to refine and enlarge his own command language through the use of macros. The user has the power to define a macro simply by giving an example of its use.

Sympathetic Parsing

The CLP is built to parse forgivingly. A pattern recognizer is built into the analysis routines which attempts to match the user input even when it does not fit the expected language form exactly. This can be done because the CLP knows more about the expected dialogue than just a parse specification.

The CLP's Service Improves with Use

The CLP maintains a lexicon for each user. It contains the user's vocabulary identified by data type. The CLP uses this lexicon to recognize incompletely spelled and abbreviated words thus saving the more experienced user much typing. The macros defined by the user are also stored. In conjunction with the User Monitor and the Tutor, records are kept of the commands known to the user and those with which he has trouble. The more interaction there is between a user and the CLP, the more information it has about his vocabulary and style and the better it is able to serve him.

In summary, the CLP provides a consistent, syntax-directed, user-oriented interface between the user and the rest of the message service.

2. THE SERVICE COMPONENTS AND HOW THEY ARE CONNECTED

The CLP, the central exchange and data base, is at the heart of the information flow paths of the military message service. The message processing service consists of the following components.

USER

The user at his terminal is, of course, the service's client.

USER/AGENT INTERFACE

Input Interface (II)

The Input Interface is the module directly responsive to the user's keystrokes. It provides immediate feedback to the user and allows input error correction.

Output Interface or Screen Control Module (SCM)

The Screen Control Module controls the content and format of the user's terminal screen. It provides subscreens or windows which other modules may use to present results to the user.

AGENT

Command Language Processor (CLP)

The CLP interprets the user's requests to the rest of the service. It also provides assistance to the user about language forms, commands available, current status and other linguistic and structural matters.

Tutor

The Tutor instructs the user. It provides more detailed assistance about the command language. It also teaches about the services provided by the Functional Modules. It provides explanations, examples and exercises.

User Monitor (UM)

The User Monitor has two tasks. First, through an analysis of the user's preliminary profile, the User Monitor determines an initial command language form. Then, while the user is using the service, the User Monitor analyzes the user's performance and, through the Tutor, suggests ways for the user to refine his command language to help. The User Monitor also records various users' degrees of success with differing command language strategies.

AGENT/FUNCTIONS INTERFACE

Function Definitions

There is a specific, well-defined form in which the functions are expected to describe to the Agent the inputs they expect and the services they perform.

Execution Interface

The CLP in conjunction with the message service Executive [6] activates Functional Modules to perform functions in response to user requests.

FUNCTIONAL MODULES

The Functional Modules do the message processing work of the service.

The remainder of this section first takes a tour of the information paths through the service. Then each of the separate CLP modules is described in more detail. Figure 3 shows how the modules are interconnected along an information flow grid.

The normal path of information flow for one command is:

User / II / CLP / FM / CLP / SCM / User

The user types a command; the Input Interface permits the user to edit it, then passes it on to the CLP; the CLP interprets it for the Functional Modules; the Functional Modules execute it and pass the results back to the CLP; the CLP records the results and passes them on to the Screen Control Module; the Screen Control Module shows them to the user. Then the user types another command.

There are a number of subcycles, detours and eddies which interpose themselves along the main flow. This section discusses the path in more detail, both the main route and the winding subpaths.

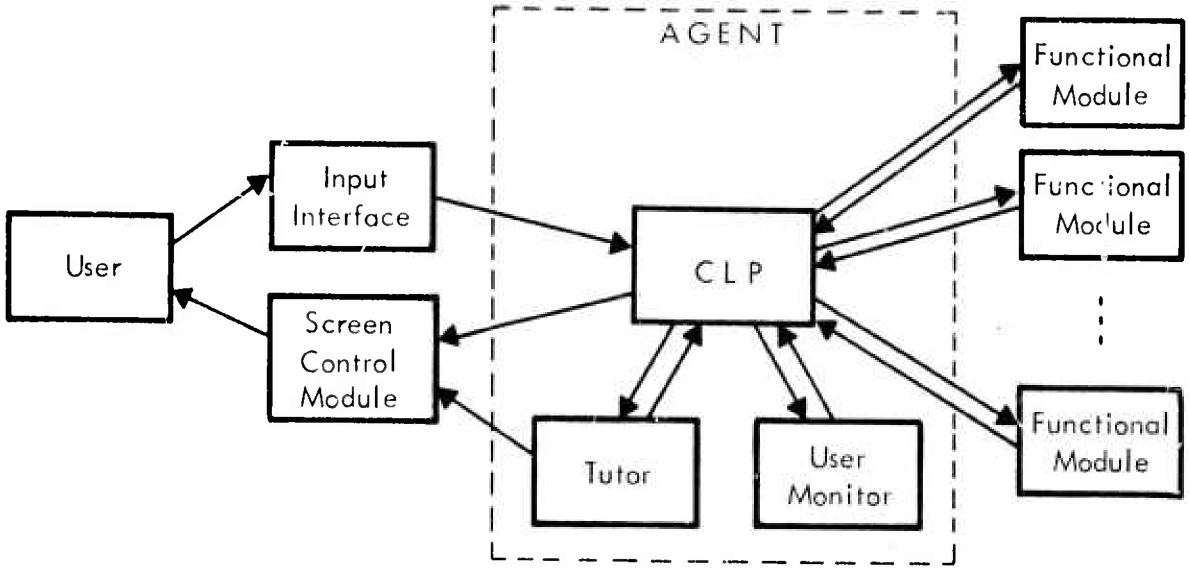


Figure 3. Information flow paths

USER

The user is assumed to be seated at the terminal. Before him, the terminal screen contains information resulting from his previous interactions. The terminal has a cursor on the screen. The cursor is positioned in such a way that the user sees the context in which he is currently working.

USER/AGENT INTERFACE

There are two components to the user/Agent interface: one each for input and output.

User/Input Interface Loop: Input Correction

Perhaps while entering his command the user mistypes one or more characters. Or, perhaps, he simply decides to change the command after entering part of it. Facilities are available to him to change his input. The Input Interface is an editor with a small, straightforward set of editing operations. Through these, the user has the ability to modify and edit his input. He may move the cursor around in the text, overtype mistyped characters, delete portions which he no longer wants, or insert new text. The commands supported by the Input Interface are a subset of those provided by the main service editor [7]. The details of the Input Interface differ from terminal to terminal.

The Screen Control Module

The information flow path between the user and the Input Interface alluded to above was not completely described. Omitted was any discussion of how the Input Interface communicates with the user. In fact, a communication from the service to the user is controlled by the Screen Control Module.

Whenever a service component wishes to send information to the user it does so through a window on the user's terminal display maintained by the Screen Control Module. All such service components request windows from the Screen Control Module, and the Screen Control Module is responsible for displaying the information contained in those windows onto the user's screen.

The user and the Input Interface communicate through a special window called the user input window. This window is the only one into which the user types. Characters typed by the user and correction functions performed by the Input Interface are displayed as corrected text in this user input window. Thus the user/II cycle is in fact a user/II/SCM cycle.

Whenever another service component of the service wishes to communicate with the user, it too requests a window from the Screen Control Module. Often it is important to display a context into which the user is to enter data. The Screen Control Module provides a mechanism to show the user as typing into a display window, still keeping the Input Interface correctional facilities available.

AGENT

The Agent itself consists of three components: the CLP to do the linguistic work; the Tutor to provide direct assistance to the user; and the User Monitor to evaluate the user's performance and make suggestions on that basis.

The CLP: Entering a Command

The CLP processes the user's (Input Interface corrected) input and attempts to recognize it as a command. The CLP works with the user, if necessary, to elicit an executable command. There are a number of services which the CLP provides to the user toward this end. The CLP can review the terminal session to date with the user to help him get his bearings. The review is not a recital of all commands executed, but a statement of the interaction contexts through which the user has passed. In reviewing the current context, the CLP provides the user with more details as to his current status.

In addition, the CLP helps the user with particulars about the current command. If the user is lost about what commands he may enter, the CLP can display the commands directly available, considering his recent inputs. Or, if the user is attempting to enter a command and is having difficulty, the CLP provides assistance.

It is possible that the user typed what he intended (or through a user/II cycle corrected the keystroke errors he made), but that the command he typed was not complete. Perhaps he left out a required parameter. In this case the CLP cannot pass the command on to the Functional Module, but must return to the user and request the unspecified argument. The CLP has a strong facility for aiding a user to build a command. The CLP is built to interact with the user in terms familiar to him. If, for example, the user leaves out the name of the recipient in a command such as "Send document ABC" (to Captain Smith), the CLP would request the user to "please enter the recipient (a person)".

The CLP may be run in a mode in which it prompts the user for every new input. In this mode, the user is fed a menu of possible inputs or input descriptions. Whenever he gets stuck, the CLP prompts him again with the next set of possibilities. Even when the user has completed a command or set of commands, the CLP keeps up with the context in which he is working and can prompt him on which commands are available and which directions his session can take next.

During each of the interaction subcycles between the user and the CLP, the user has the Input Interface correction facilities available to him. Thus the cycle is user/II/CLP/SCM.

Besides assisting the user when he does not know what is wanted of him, the CLP also analyzes the user's input to see that it is in the desired form. If one of the arguments of a command should be a date, but the user does not type a recognizable date (or a date reference such as now, today, this morning, etc.) the CLP reminds the user of what is needed. To say that an argument must be a certain type of entity such as a date is to say that the argument is of a particular data type. The appendix lists and describes the initial set of data types supported by the CLP.

The Tutor: When the User Knows He Needs Help

The powerful, sympathetic and understanding assistance available through the CLP notwithstanding, it is still possible that the user may need more help. In that case, he may request a tutorial session from the Tutor component.

The user has the facility to ask for help by pressing a special key, or simply by typing "help." When he does so, the Tutor surveys the user's recent transactions and tries to provide him with more detailed assistance than was available through the CLP. The Tutor might, for example, explain the function of a command; it might describe the meanings of the various arguments required by the command and the effect of altering them. The Tutor might set up an exercise for the user allowing him to experiment safely before attempting a command on his actual data. The exercise would be executed by the CLP and the Functional Modules in a special mode. It would operate on real data (so that the user has a real experience) but the execution would be protected so that the user does not do anything untoward.

The user's interactions with the Tutor all pass through the Input Interface and hence have the Input Interface correction features available. They also pass through the CLP. The CLP parses the user's input for the Tutor. Thus all the power of the CLP language recognizer is available to aid the user/Tutor interaction. The output, as usual, goes from the Tutor to the user through a window supplied by the Screen Control Module. The learning interaction cycle, then, becomes user/II/CLP/Tutor/SCM.

The User Monitor (UM): When the User Doesn't Know He Needs Help

There may be situations when it would be good for the user to have a session with the Tutor, but when the user does not request help. Perhaps he does not know, or has forgotten, that help is available; perhaps he is just confused or lost. Alternatively, perhaps he is doing inefficiently a task which could be performed far more efficiently if only he knew how. It is the job of the User Monitor, in conjunction with the CLP, to recognize such situations and suggest a tutorial session to the user.

There are three distinct functions performed by the User Monitor. The first two are performed off line. When the user is not in execution, the User Monitor makes a statistical analysis of the user's dialogue looking for (a) inefficient dialogue and (b) recurrent dialogue sequences. The User Monitor selects a remedial strategy if any instances of either are found. It requests that the Tutor propose that the user implement that strategy the next time he logs back onto the service. The third User Monitor service is performed on line. If the CLP has tried all its tricks with the user and the communication is still garbled, the CLP activates the User Monitor. The User Monitor in turn makes note of the situation and then proposes to the Tutor that it provide help to the user. If the user accepts the Tutor's offer of help, the two then interact in much the same way as described in the preceding section.

In this case the interaction cycle is User/II/CLP/UM/Tutor/SCM.

AGENT/SERVICE FUNCTIONS INTERFACE

There are two levels on which the Agent and the Functional Modules interface. The Agent must have a static, a priori definition of the functions in order to perform its basic task of providing an interface to the functions for the user. Second, during execution, the Agent interfaces dynamically with the Functional Modules as it directs the execution of the user's commands.

Function Definitions

The function definitions permit the CLP to transform the user's input into executable function calls to the services. These are discussed in more detail in Part 5.

Execution Interface

Functional Execution. Presumably the user will tire of playing with the Input Interface, the CLP and the Tutor and will produce an executable demand. At that time, the CLP passes the command on to the appropriate Functional Module(s) and makes notes to itself that a command is in execution.

The CLP is still available to the user. The user may, within the bounds of the restrictions noted in the next paragraph, initiate one Functional Module after another and the CLP keeps coming back for more. There is never any time when the CLP will not respond to the user. Thus the service is always alive to the user and able to interact.

There are, of course, restrictions. The Functional Modules themselves may not permit more than one command to be executing at one time. Commands to the editor are often strictly sequential. However, imagine the situation in which the user enters a command to a service which takes a long time to execute. Rather than waiting for completion, the user might wish to construct the commands to follow the one executing. The CLP is available and the user may interact with the CLP just as if the executing command had completed (as long as he doesn't need to refer to the data from the executing command). The CLP interacts with the user to produce well formed commands. When the Functional Module is free, the CLP feeds it the commands already generated by the user. The user, then, is not unduly restricted by a slowly executing command. If these follow-up commands are to new Functional Modules, then they may execute in parallel.

Whenever a Functional Module completes its execution of a command, it returns its results to the CLP via a window specification. If appropriate, the CLP passes that window on to the Screen Control Module for display to the user. In some cases, discussed next, the Functional Module response is not passed along directly to the user.

If the Functional Module is unable or unwilling to complete execution of the command because the command was misspecified at a level invisible to the CLP, the Functional Module would return the command to the CLP requesting clarification. Suppose, for example, an argument was of the appropriate data type and satisfied all the semantic tests available to the CLP, but still was not appropriate for the command. The Functional Module returns to the CLP, and the CLP notifies the User Monitor, which in turn notifies the Tutor of the error. The Tutor explains the problem to the user who may decide to enter a modified argument or, perhaps, to try a different command. In this case the cycle is user/II/CLP/FM/CLP/UM/Tutor/SCM.

If the Functional Module encounters an error condition, it informs the CLP of the difficulty. The CLP again informs the User Monitor, which activates the Tutor for an appropriate explanation of the error. The user is told of the problem and has the option of attempting to fix it (if that is within his power) or of going on to other work. The interaction cycle is the same as directly above.

The user will sometimes enter an executable command which the Functional Module will complete and which results in data for the user. In that case (really, the most common case) the cycle is user/II/CLP/FM/CLP/SCM.

In some cases, the command that the user enters is, in reality, a macro command. (The macro may be defined by the user or the service language designer or as a result of a suggestion from the User Monitor.) Then the CLP analyzes the command and causes the Functional Modules involved to execute the primitive subcommands which compose the macro command. In these situations there may be numerous CLP/FM/CLP interactions before the result is sent out to the user.

Data Base. The CLP keeps a data base of session-level data. In many cases, the CLP passes that data to service Functional Modules for them to process. In some situations, the service Functional Modules return data to the CLP for continued storage. Sometimes, the service Functional Modules work on the data in the CLP's memory space.

All service Functional Modules are welcome to do likewise. A function is especially encouraged to use the CLP's data base for storage of its own data when that information is something about which the user may know how to ask. Thus, rather than requiring the CLP to reactivate the service to respond to a user's question concerning the current value of some variable, the CLP can look up the answer directly.

FUNCTIONAL MODULES

The Functional Modules do the data processing, data transmission and computational work of the service. Each Functional Module has its own specification.

3. COMPONENT FUNCTIONAL DESCRIPTIONS

In the previous section we visited the various components of the service and saw briefly what each could do. Here we provide a more detailed functional description of three of them: The Input Interface, Screen Control Module, and CLP.

INPUT INTERFACE (II)

The Input Interface has the triple job of 1) echoing the user's keystrokes, 2) editing his input string and 3) passing along the completed input string to the CLP.

Echoing

The Input Interface has facilities to operate in either half duplex or full duplex mode. In half duplex mode, the Input Interface is activated only when the user types a key which requires either a) an Input Interface editing response or b) that the Input Interface pass the input string along to the CLP. In half duplex mode it is assumed that the terminal displays to the user the keys which he has typed.

In full duplex mode the Input Interface echoes every character in addition to peforming its editing and the CLP-intermediary jobs.

Editing

The Input Interface's editing commands are all activated by single keystrokes. There are two classes of editing commands. The main class involves cursor motion and variations on cursor motion. The second class of commands provides keystrokes to delimit new text.

Cursor Motion. Each time the user hits one of the cursor motion keys, the cursor moves one unit. With the simple motion commands, the user has easy access to any part of his input text. The actual commands are discussed below.

Text Modification. Once the user has moved the cursor to the position in the text where he wishes to make a change, he needs facilities actually to change the text.

- Text Replacement
  If the user types new text over old, the new text replaces the old on a character-by-character basis. This facility is most useful for mistyped characters. The user simply types the correct character over the wrong one. If the user mistypes a word, he may back up to the start of the word with one cursor step.

Then he may continue typing from that word as if the error never occurred. Successive characters will eliminate the incorrect word.
For the user to eliminate an extra character typed by mistake, he need only step the cursor over it in the erase mode and the character is deleted.

- o Text Insertion
  Sometimes the user wishes to insert new text into already existing text. There are two control keys which do that for him:

  Begin Text Insertion

  End Text Insertion

  Any text typed between these two keys is inserted at the current cursor position for insertion into a previously entered character string.

Cursor Motion Commands

There are control keystrokes to move the cursor either backwards or forwards in the current input character string. It is possible to move the cursor by larger or smaller steps. It is possible to have the cursor erase or not erase text it passes over.

In effect, there are three orthogonal modes which characterize the way the cursor can move.

Step unit. The unit of motion may be

- o a character
- o a lexical unit, (i.e., up to the next punctuation character sequence or space)
- o a line.

Direction of motion. The cursor might be moved backwards or forwards in the text.

Erase or Do not erase. The cursor passes over text. In the erase mode, that text is erased; out of the erase mode, it is kept. The erase mode is fleeting. It is maintained for only one cursor movement. The user is protected from accidentally erasing correct text after entering the erase mode to erase an error.

The actual keys assigned to specific functions are determined by the physical terminal used. Thus the function keys are named but not assigned specific character codes in the table below.

KEY	*FUNCTION*
erase	Enter the erase mode. (Enter non-erase mode after the next action.)
reverse	Enter backward mode. This is the base mode. The Input Interface is in backward mode at the start of each new input.
forward	Enter forward mode.
move	Step the cursor in the mode direction by one character (and erase that character if in erase mode).
move word	Step the cursor by one word, i.e., to the next punctuation character sequence in the mode direction (and erase all characters passed over if in erase mode).
move line	Step the cursor to the end of the current line in the mode direction (and erase all characters passed over if in erase mode). If the cursor is at the end of the current line, step, in the mode direction, to the end of the next line.

. The Input Interface editing functions are a subset of those provided by the main message processing service editor. The editor and all other text manipulation services are upward-compatible from the Input Interface. This service-wide consistency provides uniformity and ease of use throughout.

THE SCREEN CONTROL MODULE (SCM)

The Screen Control Module provides no services directly to the user. It is invisible to the user, but it provides essential services to other components of the service. As the Input Interface helps the user put his communication together to send to the CLP, the Screen Control Module provides facilities for service components to put their communication together to send to the user.

The Screen Control Module has sole and complete responsibility for everything displayed by the service on the user's terminal screen. When a service component wishes to display something to the user it requests a subscreen, or window, from the Screen Control Module. The service component writes into that window through text manipulation facilities provided by the Screen Control Module. The Screen Control Module displays the window to the user Even the Input Interface communicates with the user through a Screen Control Module window.

Window Specification

A window as defined by the Screen Control Module has the following elements:

Heading
A text string always displayed at the top of the window. The heading may be null.

Footing
A text string always displayed at the bottom of the window. The footing may be null.

Window Type
There are two window types: fixed-sized and scrolling. A fixed-size window displays a fixed number of text lines. A scrolling window displays as many text lines as space permits, and allow user-controlled scrolling of additional text.

Window Size
The number of text lines to be displayed (the minimum number if the window is scrolling).

Priority
The priority of this window with respect to other windows. If there is not enough room on the screen to display all windows simultaneously, those with the highest priority are displayed.

Window-ID
An identification for reference to the window.

Text
A string of character codes and special function codes. The character codes include formatting characters; the function codes indicate pseudo-cursor positions and pseudo-macro calls.

Text Pointer
A pointer into the text.

Brightness
The level of brightness at which the window is displayed.

Blinking
On/Off. If on, the window is made to blink.

Current Pseudo-Cursor
A pointer into a text string indicating which is the current pseudo-cursor position.

Overlay

A window ID. The named window, if any, is overlaid at the current pseudo-cursor position.

The Text String and Use of the Pseudo-Cursor

The text string element of a window has special functional qualities of its own. It may include special characters to mark pseudo-cursor positions. These are used to aid interactions between the module defining the window and the user. It is possible for a module to request that a certain window be displayed over its own window at indicated pseudo-cursor positions. When the overlaid window is the user input window, the user is given the impression that he is typing directly into the window of the underlying module.

Imagine a service with a form for the user to fill out. The service displays the form on the screen through a window, instructs the user to enter the first element, and through the pseudo-cursor mechanism, asks the Screen Control Module to display the user's input at the appropriate place in the form. The user will see the real cursor blinking at the spot indicated by the pseudo-cursor.

The user types the form element. He has full use of the Input Interface correctional features. When he completes his entry, the service inspects it. If valid, the service writes it into the same position on the form where the user typed it. The service then requests that the user input window be displayed at the next form entry point.

Window Manipulation Facilities

The preceding alluded to the Screen Control Module commands to manipulate windows. These commands are used by other components of the service in manipulating windows. They are not used by the user. All commands return a value and/or a success/failure indicator.

The Set commands

For each element of the window, there is a command to define or redefine it.

Create window

This command creates a window and returns its ID.

Delete window

This command deletes a window

Advance pseudo-cursor

Change the pseudo-cursor indicator to point to the next pseudo-cursor position in the text.

Text editing commands

All the Input Interface editing commands are available. They operate on the text pointer as cursor.

Other text editing commands

Other text editing commands are available as needed.

Overlay window

This command causes the indicated window to be overlaid at the current pseudo-cursor position in the base window, and hence displayed there to the user.

The Pseudo-Macros

One other very important feature provided by the Screen Control Module permits the other modules to communicate with the user in the user's own terms.

The Screen Control Module provides a limited text pseudo-macro expansion facility which it applies to text in windows. That facility permits other modules to specify data in a canonical form; the Screen Control Module will expand that canonical form to fit the input/output language form in use at the time by the user. See Part 5 for more discussion of language forms.

COMMAND LANGUAGE PROCESSOR (CLP)

The CLP transforms the user's input into executable commands. Thus its basic job is that of a language processor.

The CLP is able to recognize a number of different language forms. It transforms those language forms into canonical function calls. Part 5 discusses these forms more fully.

The CLP is also able to deal with a number of predefined data types. The appendix lists these data types.

The CLP provides a number of special services to the user. Part 4 discusses these services in more detail.

It is the goal of this subsection to provide a functional overview of the CLP's operations.

The Dialogue Grammar

The CLP interacts with the user by parsing his input with respect to a dialogue grammar. Imagine a grammar with a start symbol, or distinguished symbol (i.e., data type) of <terminal session> and with a production rule:

<terminal session> ::= <log-on><interactions><log-off>

Each of these is further broken down until an entire dialogue is specifiable. Such a grammar, if used to generate a dialogue, would produce the contributions of both members of the dialogue--not just those of one speaker. The grammar includes the CLP's interactions as well as the user's. The CLP operates by analyzing the user's input with respect to such a grammar. Some of the rules of the grammar have semantic components. Of these, some have effects internal to the CLP. Some of them activate Functional Modules. Some provide direct aid to the user and some activate the Tutor for more detailed help. In effect, the entire service is syntax-driven from the CLP.

The Session String

The CLP keeps track of the current state of the parse in a syntax graph. The path through the syntax graph from the starting point to the current input which is at the highest level in the parse structure is called the session string. The session string is the CLP's record of the user's sequence of actions. There is no complete parse of the input string until the user logs out. Yet, even though there is no complete parse, there are actions. Some of the rules which apply have semantic components which execute when they apply. These rules, of course, are relatively high-level rules and apply only when the input has been sufficiently clear. This execution mechanism is a valuable generalization of that of looking only at small segments of the input--i.e., "the commands"--and executing them when analyzable.

Not all the rules in the CLP's dialogue grammar are context-free. The session string therefore changes from time to time. As in a general production system, some rules use the session string as a scratch pad. This scratch pad feature is particularly useful to closely interactive subservices such as the Tutor and the CLP's command completion aids.

The session string is not composed of impenetrable, atomic symbols (such as the symbols used in formal production systems). Instead, the elements of the session string are blocks called syntactic units. They are identified by data type to the session string. They have general attribute/value storage mechanisms. Thus, as a command is analyzed, the block representing the command contains pointers to lower-level blocks representing arguments to the command.

An Example

Here is an example of how the CLP uses the session string to aid the user. The example arbitrarily uses a particular language form, i.e., positional functional notation. In general, comments from the CLP and the Tutor are in the same language form as that in use at the moment by the user. When a section of the session string is shown (indicated with an * at the far left), each symbol represents one syntactic unit in all cases but one. The "menu" syntactic unit consists of everything delimited by the angles. The data type of that syntactic unit is menu; the other information in that syntactic unit is stored as attribute value pairs.

Suppose the user types

    *   f(a,?

The CLP interprets that input (the question mark, in particular) as asking for a list (menu) of possible second arguments.

There is a menu rule which is activated by the ? symbol. That rule looks back in the string to determine what the user is currently doing. Then it:

- o deletes the ? from the string;
- o makes note (for the User Monitor) that the user requested a menu at this point;
- o presents a menu to the user;
- o inserts the menu into the string.

Assume that the possible second arguments are XXX, YYY and ZZZ. After application of the menu rule, the session string would appear

    *   f(a, <menu, 1 XXX, 2 YYY, 3 ZZZ>

At the same time, the semantic component of the rule sends a message to the user. It reflects the new state of the session string, but is put in terms the user understands. On the screen the message displays the menu and instructs the user to make a selection. To the user the message appears

MENU
1. XXX
2. YYY
3. ZZZ

Please type 1, 2, or 3 to make your selection.

Perhaps the user types a 2. Then the string becomes

    *   f(a, <menu, 1 XXX, 2 YYY, 3 ZZZ> 2

The next rule to apply is the menu selection rule. It is activated by the menu element and it picks the second choice on the menu as the user has asked. The rule then deletes the menu from the string, which now appears

* f(a, YYY,

The user may continue.

CLP Activations

The CLP is event-driven. Whenever the user completes an input, the Input Interface passes it on to the CLP. That action causes the CLP to resume execution. The CLP appends that new input to the session string. It then continues its parsing algorithm on the newly expanded session string.

Whenever an active Functional Module completes its current task, the CLP is also notified. It resumes execution, and in this case modifies the session string according to the results produced by the Functional Module.

The CLP routes the results of Functional Module activations to differing destinations. In the standard case, the CLP passes the results on to the user. If, on the other hand, the Functional Module signals an abnormal result, the CLP notifies the User Monitor which, in turn, activates the Tutor to explain the abnormality to the user.

If the Functional Module execution had been performed as an exercise proposed by the Tutor, the CLP passes the results back to the Tutor for display to the user and whatever further explanation the Tutor determines is appropriate.

Contexts

Having examined the possible information flows for a single command, it is worthwhile to consider briefly how the CLP deals with interactions on a longer time scale.

The CLP supports a context mechanism which organizes commands into related groups. The context notions are explained in more detail in Section 4.

o Through the use of contexts the user may communicate with the Agent in an abbreviated manner. It is not always necessary to express a command in complete detail; the context makes meaningful what might otherwise be a meaningless partial command.

o Contexts are structured both horizontally and vertically. The user is free to move in any direction. He may move from one context to an adjacent context. He may also skip over contexts. The user switches contexts by entering a command which is not in his current context. The CLP searches out the closest context which does contain that command. When the user does jump a number of contexts, the CLP in effect inserts dummy commands which take the flow of the interaction through those contexts. Thus the user has all the advantages of structured interaction without the disadvantages of excessive detail.

o Contexts permits the CLP to provide a pronominal reference service. Pronouns do not refer back into contexts which have been completed [8]. Thus contexts which the user has left are not examined during the search for pronoun antecedents.

o Contexts permit the CLP to prompt the user intelligently about his open options. They also allow the CLP to review the user's status with him. The Tutor uses the contexts for similar purposes.

Summary

The CLP is an event-driven language processor which interacts with the user in accordance with a dialogue grammar. The grammar includes the notion of interaction contexts which permit more natural and free-flowing activities by the user.

4. SESSION CONTROL STRUCTURES AND CLP SERVICES

The CLP provides a number of aids to the user. Before describing the specific user assistance features, further discussion of the CLP context mechanism is presented.

CONTEXTS

During a terminal session, the user is always operating in some interaction context. As he performs various operations, he moves from context to context. The context feature permits the command language to be much simpler than it might otherwise be if each command had to be completely self-contained. On the other hand, the contexts do not compartmentalize the various subcomponents of the service as is the case with most vertically structured services.

A context is simply a collection of commands. A context has a name and some descriptive material attached to it. The main function of a context is to group commands together and show how they relate to other commands.

Each context is made of of a) pointers to the commands it contains and b) pointers to other contexts. Besides contexts having links from one to another, commands too may have links from themselves to contexts. Commands are linked to contexts in two ways and contexts are linked to each other in three ways. Here we mention the types of links available between contexts considered as collections of commands. These links grow out of the links between groups of Functional Module functions discussed in Part 5.

Command/Context Links

Command link type 1. Each command is linked to the context in which it is found. That is, a context points to all the commands which it contains, and

Command link type 2. Some commands create subcontexts. Each command which creates a subcontext has a pointer to that context.

For example, some Tutor commands create contexts in which the user is expected to respond. Each such command points to the context which has the rules to deal with that expected response.

Context/Context Links

Context link type 1. Each context which is a subcontext in the sense of command link 2 is linked to the context containing the command which activates it as a subcontext. That is, each context links to its supercontext.

Context links types 2 and 3. In some cases, interaction with the user moves step by step from context to context in some well-defined, finite state manner. Contexts may have forward and backward links which point from themselves as a current context to a next context and from the current context back to a previous context. The log-on commands are a context which refer to the general Agent commands as a forward context. The general Agent commands refer to the log-off rules as a forward context.

The CLP uses the context mechanism to recognize user commands. The user is always in some current context. From that starting point, the CLP cycles through related contexts searching for command formats which match the user input. When a command is found which matches, the search is terminated. If no command is found, the CLP fails in its search and takes appropriate action.

This context mechanism provides a formal structure to interactions without trapping the user in stacks and pushdown stores. The pushdown mechanism is available if the user chooses to make use of it. But if he wishes to switch context, he is completely free to do so, just as long as the vehicle he uses is not meaningful in a context closer to his current one than he had intended. The mechanism also allows different contexts to use the same commands to mean different things. Since the CLP stops at the first successful match, the problem of ambiguity does not arise. The only constraint imposed is that each context must be internally free from ambiguity.

SERVICES PROVIDED BY THE CLP

The CLP provides a number of services to the user.

Prompts

When the prompt mode is enabled, the CLP will respond to the user's request for a prompt by indicating what possibilities are available next. Again, the form of the CLP's response is dependent upon the particular language form currently in use. Depending upon the immediately preceding input, there are a number of different types of next inputs.

- If the user is in the middle of entering a command, the CLP prompts the user with the name and description of the next command component expected. If the argument has associated with it a short list of possible values, the CLP displays that list.

- If the user has just completed a command and is in a specific interaction context, the CLP prompts the user with the names and descriptions of other commands available in that context.

- If the user is in a position to switch from one interaction context to another, the CLP prompts the user with the names and descriptions of other interaction contexts. The CLP also lists the main commands in each of the contexts mentioned.

- If the user is in need of more help, the CLP shows the user how he can ask for more detailed aid either from the CLP or from the Tutor.

Often, the CLP responds to a prompt request with information from more than one of these categories.

Aborts

The user may request that the current line of pursuit be abandoned. As in the situation with prompts, the CLP's response depends on the user's current needs.

The user has the option of specifying the scope of the abort. Some such are: current command, previous command, current context. If he does provide the explicit specification, the CLP aborts as requested and resumes interaction in the previous context. If the user does not specify the abort argument, the CLP makes a guess and then requests a confirmation from the user.

The CLP's guess is also context-dependent. If the user had just completed a command before entering the abort request, the CLP guesses that the abort applies to that command. If the user is in the midst of entering a command when he requests an abort, the CLP guesses that the abort applies to the currently incomplete command.

Whenever the user requests an abort of one of his commands which has begun but not finished execution, the CLP passes that request along to the executing Functional Module. It is the job of the Functional Module to accept abort requests and to abort cleanly.

Undo

The user may request that a previous operation on his part be undone. Undo may apply to a command or it may apply to his entry of a command argument during an interaction with the CLP.

If the user requests that a command argument specification be undone, the CLP simply deletes that specification from the command under construction.

The Functional Modules in the military message service are constructed in such a way that they can undo commands to the greatest extent possible. When the user requests that a command be undone, the CLP passes that request along to the Functional Module which executed the command. It is up to the Functional Module actually to perform the undo. The CLP passes the Functional Modules' response back to the user after translating it into terms familiar to the user.

Part of the description defining Functional Module functions to the CLP is the Undo flag (see Part 5). It indicates whether or not the Functional Module is capable of undoing the function. When the user requests a function which the Functional Module cannot undo, the CLP may warn the user and request a special confirmation from the user before executing the function.

Where-Am-I and Review

At any point, the user may request a summary of his current command and interaction context. When the user asks the CLP "Where Am I?" the CLP reviews with the user his current hierarchy of open contexts. In addition, if the user is entering a command, the CLP also displays that command, to the extent to which it is defined.

The user may also request that the CLP review with him the work that he has done. For a general review, the CLP reports to the user all commands executed in the highest, i.e., the general Agent, context plus other commands in currently open contexts. If any of those commands initiated some interaction in a closed, lower-level context, the CLP reports the fact of that lower-level interaction, but does not automatically review it. In effect, the CLP translates and displays the session string to the user.

The user may, however, request a review of his activities in any of the lower-level contexts mentioned above. In that case, the CLP reviews his actions there and reports, but again does not give details, of any still lower-level contexts entered. Through this mechanism, the user has the means to see any of the work he has done. At the same time, he is not overwhelmed with unwanted detail.

Execution from a File

The CLP has facilities to accept input from a file. When operating in this mode, the CLP assumes that there is a user at the terminal. If there are any problems encountered in the file material, the CLP requests clarification from the user at the terminal.

Thus the CLP accepts input from the file, but interacts with the user at the terminal. It is possible, then, to write a program (i.e., sequence of commands), store it on a file, then execute it while accepting input from the terminal. The CLP has defined a special data type <terminal-input> which can be used in file programs whenever it is desired to accept a parameter from the terminal.

The terminal user is never locked out. The CLP is always open to accepting input from the terminal. Anything the user types at the terminal takes precedence over commands from the file. It is possible, of course, for the terminal user to request that execution from the file terminate. Normally, the CLP displays the file to the user. He can watch the program as it executes and stop it whenever he chooses.

The editor acts as intermediary when the CLP is executing from a file. When the user gives a command to execute from a file, he specifies where in the file execution should begin. The CLP passes on that specification to the editor, which in turn feeds the CLP text from the file. The user may be editing the file at the time. In that case a command such as execute from here (the current pointer position into the file) makes sense to the editor. In any event, whatever the specification, it is a text specification. The user already knows how to express these to the editor. The editor need not perform any operations on the file as it is passed to the CLP. Any corrector functions entered by the user will already have been performed before the text was stored in the file.

Identifier Lexicon, Abbreviations, and Recognition

The Lexicon. For each user, the CLP mantains an individual lexicon of identifiers. Whenever the user issues a command, the parameters used in that command are stored in that user's identifier lexicon.

The lexicon records how and when each identifier was used. Each time an identifier is used, the data type of the parameter for which it was used is stored along with the date of that use. A count is kept of the number of times an identifier is used as a particular data type. The most recent date of use is the one date maintained for each data type usage.

The CLP uses this lexicon to aid its command recognition algorithm. When the user first enters the service, his personal name space is unfamiliar to the CLP. As he uses the service, however, the CLP gathers a collection of the names which he uses and the ways in which he uses them. These lexicon entries do not restrict the user from using a name in new or in multiple ways; the CLP does not require that the user conform to his previous usage. Rather, the CLP uses the lexicon to help produce an initial analysis of the user's input command and so to respond faster to the user.

Abbreviations. The CLP permits the user to abbreviate identifiers. The user may terminate a word-sized unit by a special abbreviation character. The CLP then attempts to find an identifier from the identifier lexicon which abbreviates to those characters. Using the simplest abbreviation algorithm, the CLP looks for identifiers which begin with the characters typed. More complex pattern matching abbreviation algorithms may be available if appropriate.

When an abbreviation expansion seems correct, the CLP requests a confirmation from the user. If confirmed, the abbreviation is entered into the lexicon with a link to its expansion. The next time the abbreviation is used, the CLP finds it directly in the lexicon. Facilities are also available to permit the user to define abbreviations explicitly --as well as implicitly by use.

Recognition and Prompting. The CLP provides a special type of abbreviation service called recognition. When the user hits a special, designated recognition key during command input, the CLP is activated. The CLP analyzes the input provided and looks up all abbreviations in the input. Then the CLP substitutes the complete lexical unit for the abbreviation and adds a prompt for the next expected input, if any. Then the CLP returns the expanded input string to the Input Interface for further characters. The user may perform any corrector actions he desires on the expanded string just as if he had typed it himself. When the user finishes the command, the Input Interface returns the string to the CLP for final processing.

Recognition differs from abbreviation expansion in that it is performed immediately. Abbreviations are expanded after the user has completed the entire command and submitted it to the CLP for execution. Recognition is performed for each lexical unit immediately when it is requested.

Argument Possibilities List. The CLP also makes use of the lexicon to give the user a list of the possible arguments to a command. Normally, when the user requests a prompt for a command argument, the CLP provides a description of the expected argument. For example, the CLP may inform the user that the next argument is expected to be a document name. However, if the user chooses, he may request more help from the CLP. Perhaps the user forgot the name of the document he wanted. The CLP has facilities to find all document names in the user's lexicon and to display them. This particular service is of value when the user is not lost or confused about what is needed, but has simply forgotten the specific name he wants to use.

The User Training Statistics Table and Mode Value

The User Monitor and the Tutor maintain a table which shows which of the service functions the user has been taught. The CLP uses this table to determine which input commands the user is likely to give.

There are two modes in which the CLP uses this training statistics table: permissive and restrictive. In the restrictive mode, the CLP does not execute any functions for the user which the table shows him not to know. In the permissive mode, the CLP executes functions about which the user has not been tutored. In doing so, though, the CLP requests special confirmation from the user. It also reports these executions to the User Monitor.

Macros

The CLP supports a facility for the creation of macro-commands. The macro definition feature may be used either by users or by language designers (see Part 5 below). A user might define a macro command when, for example, he finds himself executing some particular sequence of commands repeatedly. He would define a macro command to perform the entire sequence of commands at once. Sometimes the User Monitor, via the Tutor, suggests such possible macros to the user.

A language designer might define a macro command to create a more user-oriented service interface. That is, a service may be initially defined with a large number of service-dependent mini-functions. While appropriate for the structure of the service, these individual mini-functions may be of little use to a user. The language designer for the service would use the macro feature to define commands more in line with the user's interests. (See Part 5 for more details on language design.)

Defining a Macro. The macro creation facility defines a macro from an example of its use. A macro definition consists of three components.

- o the macro call form;
- o the word "means" (or some other indicator that a macro is being defined);
- o the macro definition;

To define a macro, one enters:

text-string-1 means text-string-2

where text-string-1 is an example of the macro call form and text-string-2 is the intended meaning.

For example, assume that a document must be "Released" before it can be "Sent." A user might define a new "Mail" command which performs both operations.

The user enters:

Mail XYZ to Captain Smith means
Release XYZ
Send Captain Smith XYZ.

Once defined, the user may use that command for any combination of documents and people--not just XYZ and Captain Smith. The effect will always be 1) to Release the document and 2) to Send the document to the named person.

Notice that the user defined the macro by giving an example of its use. He did not have to write a program using symbolic variables or special macro parameters and formats.

The CLP's Response to a Macro Definition. When the user enters a macro definition, the CLP analyzes it and then returns for confirmation to the user with its analysis of the definition. The CLP's analysis is couched in a fill-in-the-blanks format.

In the previous example, the CLP would respond:

```
You are defining a macro-command.
Mail ---------- to --------
      document      person
means
Release ----------
          document
Send ---------- to --------.
       document     person
```

If the user confirms this interpretation, the macro is stored as part of the user's personal data.

Macro Documentation. The CLP permits (in fact encourages) users to supply documentation with their macros. After a macro is defined, the CLP returns to the user and requests a brief English description of the macro's function. That documentation is stored along with the macro. It is of use to the user if he forgets his intention or to others who may use his macro. It is also used by the Tutor in explaining the macro.

Pronouns

The CLP permits the user to enter pronouns as command arguments. When the CLP encounters a pronoun, it searches back into the session for the most recent argument of the same data type which occurred in a command whose context is accessible from the current user context. It is that value which the CLP takes as the intended antecedent. Before executing the command with the implied argument, the CLP normally displays the completed command to the user for confirmation.

Session-to-Session Continuity

As indicated above, the CLP develops more and more information about the user. A lexicon is built; user-defined macros are stored; abbreviations used by the user are kept; records are maintained of the commands known to the user and those with which he has trouble. So, the more the user interacts with the CLP, the better it knows him and the better it can serve him.

The CLP relies on services from the Executive module to provide a second form of session-to-session continuity. When the user ends a session by logging off, a check-point snapshot is taken of all session level parameters. The next time the user logs back on to the message service, the CLP offers him the option of resuming his work exactly where he was when he exited previously. The user may choose to do so, in which case interactions proceed as if there had been no interruption. Or the user may choose to begin a fresh terminal session and disregard the checkpointed information.

SUMMARY

The context mechanism and the special services offered by the CLP are all aimed at providing a service environment as friendly and comfortable as possible for the service user. He is the client, the customer, and it is his convenience which the CLP and the Agent intend to serve.

5. SERVICE DESCRIPTION

The previous sections discussed how the user and Agent interact. It described how the user could move around from one context to another; how, within any context, he executes commands; and what services the CLP provides to assist him in entering those commands. This section describes how those commands are defined.

INTRODUCTION TO MULTIPLE LANGUAGE FORMS

Most services would not have a section on this level since the commands would be designed along with the service. The message service is not built in that way. There are three reasons the commands are not built into the service beforehand.

1. The military message processing service is a model for a class of services any one of which may be attached as Functional Modules to the front-end Agent. These other services are as yet undefined. Their commands also are necessarily unknown.

2. One of the goals of the Information Automation project is an analysis of various command language variants for their impact on the ultimate user. For this reason, the project has not picked just a single language form, but has permitted the implementation of a number of command languages for the same service functions. It is expected that different language forms will be variously suited to differing user populations.

3. If the experimentation of (2) yields the hypothesized results, then from the user organization's point of view too it is important not to predefine the ultimate language. Different user organizations will want differing language forms. For their individual purposes they will need to tailor the commands to their own needs so that the emphases are different.

Information Sources for Commands. A command embodies information from two different sources. First, a command must relate to the underlying service functions which its execution invokes. Second, a command reflects the linguistic and other local choices which determine its form and default structure. To describe how a command is defined we first review the functional specifications expected from the Functional Module authors. Then we discuss the range of choices available to the language designer for constructing commands which activate those functions.

Thus two different groups of people are involved in the definition of a command language.

1. The Functional Module authors who build the services and provide functional descriptions for them; and

2. The language designers who take these functional descriptions and design commands to meet the needs of their particular audience.

Once the commands are designed, they are connected to the Agent. The CLP uses them as a table-driven interface between user and Functional Module.

SERVICE FUNCTIONAL MODULE SPECIFICATION

The author of a service Functional Module is expected to provide a functional specification of his Functional Module. A functional specification consists of sets of service functions called contexts. Each context contains a number of functions. First we discuss the context specification requirements and then describe how a service module author specifies the individual functions.

Contexts

The context mechanism is a way of organizing service functions. The Functional Module writer may organize his functions into related groups. He may require that these groups be executed in some sequence. He may also list some groups as subordinate to specific functions.

That is, a service may require that the user pass through a sequence of states: state 1, state 2,..., state n. In each state the user has the option of performing some particular set of service functions. The Functional Module author groups his functions into contexts which represent these states. When the user is in context 1, he is expected to be performing one of the service functions in state 1.

Recall that the user is not restricted to performing only those service functions in his current context. He may slide from context to context and the CLP keeps up with him. The context mechanism is not a way of constraining the user. In fact, it permits the command language to be much simpler than it might otherwise be if each command had to be completely self-contained. The contexts do not rigidly compartmentalize the various sub-components of the service as is the case with most vertically structured services. For our purposes in this section, though, the contexts provide the Functional Module author a way to organize and build structure into the collection of functions he offers.

Once the Functional Module author has his functions grouped into contexts, he may indicate how these groups are linked together. Two contexts are considered linked together if the user may move from performing functions from the first to performing functions from the second.

Context Links: Hierarchical and Sequential. There are two basic ways contexts may be linked: hierarchically and sequentially. Two contexts are linked hierarchically if the user is expected to return to the first context after completing his work in the second. A command in one context with its subcommands comprising another context is an example of two contexts linked hierarchically.

Two contexts are linked sequentially if there is no requirement that the user will necessarily return from the second to the first.

Passageway Functions. These two ways of linking contexts may be further qualified. In some cases a user may pass from one context to another only by executing a specific function. In other cases, no such function passageway is required. Thus a context consisting of subcommands is entered only by executing the particular command which leads to the subcommands. Alternatively, a context consisting of the major commands of a service may be entered from the Agent context simply (in most cases) by executing one of that service's commands. In general, it is not necessary to invoke the service with an additional explicit command. Both of these examples are cases of contexts linked hierarchically.

There are also examples of sequentially linked contexts with and without function entries. Consider an editor with two function contexts: one to edit a specific document and the other to perform file manipulation operations. Each may be linked to the other sequentially (forming a cycle). To move from the file manipulation context to the document edit context, the user must somehow mention the document to be edited. He must use a function which refers to a document. So only through the use of certain functions may one enter the edit context. However, to move from the edit context to the file manipulation context requires no such mention of a particular document. The user may execute a function in the second (file manipulation) context directly without an intermediate step of entering that context.

CONTEXT DESCRIPTORS

A context as specified by a Functional Module author is specified with four components. A context descriptor might appear as follows.

Context-Name:
Sequentially-Linked-Contexts Lists,
Hierarchically-Linked-Contexts Lists,
Functions List

where

Context-Name is a unique, arbitrary, symbolic identifier.

Sequentially-Linked-Contexts List is a list of context names. Each context name may be qualified by the name of an entrance function.

Hierarchically-Linked-Contexts List is identical in form to the Sequentially-Linked-Contexts List.

Functions List is a list of function names. These are the names of those functions considered grouped together to make up this context. The function names used in qualifying the context names for lists (2) and (3) must also be in this list.

This completes the brief discussion of contexts. Next, we discuss the specifications of functions.

Functions

As far as the CLP is concerned, a Functional Module consists of functions (which may be organized into some structure through the context mechanism). The Functional Module author must describe these functions to the CLP through function descriptors.

The descriptor for a function has three parts. A function description might appear as follows:

function-name (argument-list) undo-flag

Function Name. The function name is a unique, arbitrary, symbolic identifier.

Argument List. Each function which may take arguments has an argument list. The arguments are defined to the CLP positionally. For each argument, the function descriptor provides a data type and a flag signaling whether or not the argument is required for the function to execute.

Each argument is specified by an argument descriptor:

<data-type, requirement flag>

Data Types

The specific data types supported are discussed immediately below. An argument's data type specifies the form of data expected by the function for that argument.

The Requirement Flag

It is possible for a function to provide an argument position and yet not require that the argument be supplied. (Note: To say that an argument is not required is not to say that a default value is assumed. Default values are defined below in the language design section.)

Thus the argument list is a (possibly null) sequence of ordered pairs, one for each argument. The first element of the pair is a data type name; the second element is the required/not-required signal.

Undo Flag. The undo flag is set on a function which the Functional Module is able to undo. The CLP uses the absence of this flag to warn users when they are entering commands which might result in unalterable changes.

Service Argument Data Types

The data type field in an argument descriptor specifies which data type the argument must be. This field also provides context to allow the CLP to differentiate among commands during command analysis. There are several classes of data types which are supported by the CLP. They are:

Number Item Types
Chronological Types
Personnel Types
Priority
Security Handling
Message Type
Signoff
Reviewer
Address
Recipient
Date-Time Span
Message Identifier

These data types are discussed in detail in ***Basic Functional Capabilities for a Military Message Processing Service*** [3] and in the appendix.

To summarize, Functional Module authors specify their services to the CLP by providing (1) function descriptors for all the functions which they perform and (2) context groupings and structure for those functions.

LANGUAGE DESIGN

It is the task of the language designer to provide a set of commands for the execution of Functional Module functions. The language designer has available to him (1) the service module function descriptors and context information and (2) tools provided by the CLP for designing a command language.

A command language consists of a collection of commands. The domain of the command language designer includes only the design of individual commands. The language designer has no power to specify explicitiy any intercommand structure.

There is, in fact, an intercommand structure, but that structure grows out of the context specification provided by the Functional Module author. While the language designer is then limited in the structuring power available to him, he is not restricted solely to creating commands which map one-to-one onto the service functions. A command may cause the execution of more than one service function and hence may effectively move the user from context to context. Or, the language designer may require a number of commands from the user to execute a single service function. And, of course, the language designer may map some commands one-to-one directly onto service functions.

The Commands Linguistically

To specify the commands linguistically, the language designer defines six elements:

1. The Command Language Form
2. The Vocabulary
3. The Argument Default Values
4. The Argument Visibility Values
5. Warning and Confirmation Indicators
6. The User-Driven or CLP-Driven Indicator

Each of these is discussed individually.

a. The Command Form. There are a number of command language forms supported by the CLP. For each command, the language designer selects one of these forms.

- Functional Positional Notation
  The language designer may specify commands as using functional positional notation. In this form the name of the command is followed by an optionally parenthesized list of arguments. The arguments are assigned positionally. The arguments may be separated by commas. An argument may be an expression. Arguments omitted are indicated by successive commas.

- Functional Keyword Format
  The functional keyword form is similar to the functional positional notation. Here too the command name leads to be followed by an optionally parenthesized list of arguments. Only in this case the arguments are given by argument name rather than by position.

  For example, assume that the names of the arguments of the command which sends a document are: Sender, Recipient and Document. Perhaps the command name is Send. Using the functional keyword command form, the user would enter

  Send (Document = SSD2, RECIPIENT = ARPA)

  The arguments might be given in any order. Note that the Sender argument is omitted. The CLP supplies the default value specified by the language designer, which in this case might be the user himself. Section c. below provides a discussion of default values.

o Simple English
The CLP supports a simple English type of input language. If the user is operating in this mode, he will type most of his commands as short command-like sentences. In general, in this form the command name serves as the main verb of the sentence. The other arguments complete the required case elements of the sentence. These will vary depending upon the specific function involved.

o Function Keys and Names
With this language form, the user indicates a command through the use of a single key rather than the command name.

When operating in this language form, the CLP always has displayed on the terminal screen a list of the currently available functions in the current context and a list of the available new contexts to which the user may wish to move. Preceding each function name is an indication of a keyboard key. These will in most cases be the letter beginning the name.

The user specifies which function he wishes to execute (or which new context he wishes to enter) by hitting the single key indicated. When the user hits a function key, the CLP immediately emphasizes the display of the specific function name and adds an additional display to the terminal screen, the list of input argument names. The CLP provides an emphasis to the argument currently expected. For each argument, the CLP includes still another display. The individual argument display shows either of two possibilities. In one case, the CLP displays the argument data type (e.g., for the argument "recipient", the data type is person). If the argument may be picked up from a predefined set of possibilities known to the CLP (the other case), then the CLP displays these specific possibilities along with the function keys used to select them.

Besides the function keys used to specify arguments, the CLP has available function keys to activate its regular services. There are function keys for Abort, Help, Where-Am-I, etc. There is no special prompt key, since in this mode the CLP is prompting all the time.

b. The Vocabulary. For each of the language forms indicated above, the language designer must provide the vocabulary. The vocabulary elements (some of which were alluded to already) are:

o Command Name
Each command has a command name.

- Argument Names
  Each command argument has its own name.

- Simple English
  For the simple English language form the designer must supply a lexically defined sample English sentence for each command. (The CLP analyzes that sample sentence with its simple English grammar. The result of that analysis provides the Simple English command form expected from the user.) Thus the designer must provide both the sample sentence and the vocabulary data required to parse it.

c. The Argument Default Values. For each command argument, the language designer must determine whether or not default values can be supplied. If default values are permitted, the CLP provides tools for the language designer to specify what the default values should be.

Default values fall into four classes.

1. Constants
   The simplest class of default values are the constants. The language designer may declare that a defaulted argument take a specific constant as value.

2. Global Service Values
   For some arguments, a constant default value is not sufficient. The CLP provides functions which supply information such as time of day, date, organization name and location, etc.

3. User Specific Va[illegible]
   Some default v[illegible]ould be user-specific. Thus the CLP permits the language designer to specify such values as user name, user title, user location, user security level, etc.

4. Pronominal References
   The CLP supports a limited pronominal reference facility. Sometimes, it is convenient to default references which have just been made. Thus after creating a document, the user may Send it without repeating its name by defaulting the Document Name argument in the Send command. If the language designer specifies pronominal reference as the default value of some argument, the CLP will take as the value the most recent argument of the same data type in a context which is still open.

d. The Argument Visibility Values. The CLP provides the language designer with the ability to make an argument more or less visible to the user. There are three levels of visibility:

1. Active
   If a command argument is declared actively visible, the CLP takes every opportunity to inform the user that the argument exists and may be given a value by him. If, for example, the user defaults an actively visible argument, the CLP will inform the user that the argument has been defaulted and will display its default value. The language designer should declare an argument actively visible when he believes it is important for the user to know that there is such an argument.

2. Passive
   An argument declared passively visible is not pushed on the user. Typically, beginning users are not taught about passively visible arguments initially.

   A user need not know about passively visible arguments to use the service. However, experienced users who have learned about them may manipulate these arguments to their benefit. The CLP does not prompt a user for a passively visible argument. The CLP does respond to inquiries from users about their options by displaying passively visible arguments.

   If an argument is declared passively visible, and if that command argument supplies a service function argument which the service module author has declared required, then the language designer must supply a default specification.

3. Invisible
   The language designer may declare some arguments invisible. A service function may have left open an argument option which the language designer may wish to close.

   For example, the rank and title of the user may be an argument to a service function. Yet the language designer may want to prevent the user from specifying his own rank. He may declare that argument invisible. Invisible arguments must be provided with default specifications if they are used as required arguments for Functional Module functions.

e. Warning and Confirmation Indicators. Recall that the service Functional Modules are not able to provide UNDO facilities for all functions. The language designer may wish to warn the user about commands which activate such functions. The language designer may attach a warning flag to any command. The CLP will warn the user that the command cannot be completely UNDOne before executing it.

In these cases, it is important to require a confirmation signal from the user. Sometimes in other situations too the language designer may wish to force the user to confirm commands before execution. Commands marked with a confirmation flag are not executed by the CLP until the CLP has issued a confirmation request to the user and the user has confirmed the command. All commands with warning flags from the Functional Modules are automatically marked with confirm flags.

There are cases in which it is not the entire command which should be confirmed but just an individual argument. The language designer may attach confirmation flags to arguments as well as to commands.

j. User-Driven or CLP-Driven. The language designer may indicate whether the primary mover in entering commands is the user or the CLP. If the command language is user-driven, then the CLP waits for the user to enter commands and interacts only as an assistant. If the command language is CLP-driven, then the CLP takes the initiative in prompting the user to enter commands. The CLP prompts the user by executing in a mode called Fill-in-the-Blanks.

The Fill-in-the-Blanks input mode works in conjunction with any of the other forms. The CLP provides the user with as much prompting as possible, which comes in the form of a set of blank spaces presented in context with descriptions beneath each blank space.

Let's consider the functional notation language form used in Fill-in-the-Blanks mode. If the user has not indicated a particular command to execute, the CLP prompts

```
--------------   (                    ).
   command             arguments
```

Once the user has specified the function, the CLP reprompts the argument blanks to match the specific function. Thus if the user types "Send", the CLP responds:

```
    Send (------------, -------------, --------,  ----------)
command  document      recipient      sender     priority
```

The user may then fill in or default the other arguments.

When Fill-in-the Blanks is used with Simple English, the CLP displays the service-supplied English command sentences with blanks (described underneath) for the arguments.

The Commands Semantically

To complete the specification of the commands, the language designer must indicate how the commands he defined correspond to the service functions defined by the service author.

To indicate which commands correspond to which functions, the language designer constructs a command macro. A command macro is a special kind of macro in which the call string consists of commands. (See Part 4, "Macros" for more details about how a macro is built.) For the language designer, the macros which he creates consist of two major components: the commands string and the service function string. The macro, in effect, equates the two and defines how the arguments from the commands are used to supply arguments to the functions.

Consider an example in which the commands have been defined using functional notation. Let c1, c2,...,ck be some commands and let f1, f2,...,fj be some service functions. Then the language designer might declare

c1 (a1, a2) c2 (a3)...ck (an) means
f1 (a3, an) f2 (a1, a3)...fj (a1, a2, a3, am)

The CLP takes that declaration and builds internal rules which reflect the indicated relationships. Thus if the user ever enters the command sequence c1, c2,...,ck then the CLP will execute the service functions f1, f2,...,fj with the appropriate arguments. Note that this mechanism permits the language designer to include as many or as few commands and as many or as few service functions as he wishes in any such declaration.

SUMMARY

This completes the description of how the Functional Modules and the command languages are specified. A service designed along these lines will have clean internal interfaces. At the same time it will be user-oriented and pleasurable to use.

The primary aim of the message service is to provide computation and data processing services to users who have little knowledge of computers and data processing. This document has described some of the major components of the message processing service and has shown how the CLP, the Input Interface and the Screen Control Module help achieve this important goal.

REFERENCES

1 U.S. Air Force, *ESD Study on Intra-Base Communications*, 1973.

2 Ellis, T. O., L. G. Gallenson, J. F. Heafner, and J. T. Melvin, *A Plan for Consolidation and Automation of Military Telecommunications on Oahu*, May 1973, USC/Information Sciences Institute, ISI/RR-73-12.

3 Tugender, R., D. R. Oestreicher, *Basic Functional Capabilities for a Military Message Processing Service*, ISI/RR-74-23 (in preparation).

4 Heafner, J. F., *A Methodology for Selecting and Refining Man-Computer Languages to Improve Users' Performance*, September 1974, USC/Information Sciences Institute, ISI/RR-74-21.

5 Rothenberg, J. G., *An Intelligent Tutor: On-line Documentation and Help for a Military Message Service*, ISI/RR-74-26 (in preparation).

6 Mandell, R. L., *An Executive Design to Support Military Message Processing Under TENEX*, ISI/RR-74-25 (in preparation).

7 Rothenberg, J. G., *An Editor to Support Military Message Processing Personnel*, ISI/RR-74-27 (in preparation).

8 Deutsch, B. G., *The Structure of Task Oriented Dialogues*, Stanford Research Institute Technical Note 9, SRI Project 1526, Menlo Park, California, 1974.

APPENDIX: DATA TYPES

The data types are one place within the Agent where semantic knowledge of the service is known. The knowledge of data types allows the Agent to respond intelligently to end-user inputs.

Associated with each data type is the recognized input forms, the internal format which is used by both the Agent and the functional modules, and the output forms. The data types fall into seven classes:

1. Text Items.
   A text item is a string of uninterpreted characters. The Agent is not concerned with the information contained in these items.

2. Chronological Items
   These items describe both points in time and periods of time.

3. Personnel Items
   Personnel items refer to individuals and groups of end-users. The Agent can check these items for validity.

4. Special Categories
   The special categories are small specialized vocabularies for special situations. The Agent can also check these for validity.

5. Quantities
   Quantities included cardinals, ordinals, and money amounts. Reasonableness ranges can be supplied for first-order checks by the Agent.

6. Transmission Specifications
   These are often-used compound items which specify where a message is to be delivered, along with certain delivery parameters.

7. Message Identifier
   This is a service message identifier. It refers to an actual message and can be checked by the Agent.

Each class is discussed in general below with primary emphasis on the input forms recognized for the specific items in that class. At the end of this section is an alphabetical listing of all data types and their associated internal formats and output forms. For each type there are associated long and short output formats.

APPENDIX

Text Items

Character/Word/Line/Text

A text item is simply a string of uninterpreted characters.

The start of a text item must be indicated by an unambiguous text item in the argument list. While the user is entering text there are two keys which cannot be entered into the text item: the escape key and the text item termination key. The escape key will enable user activation of macros (discussed in Part 4 above) within text items. The termination key ends the text item.

There are actually four text item data types: character, word, line, and text. Character is a single character. Word is terminated by a space, tab, or punctuation mark. Line is terminated by a carriage return or other end-of-line key. A text is terminated by a terminate key. This latter key, as with the escape key, is determined by the physical terminal design.

Chronological Data Types

Date/Time/Date-Time/Date-Time Range

The chronological data types represent dates and times and are recognized by the CLP.

Dates

The input format for dates is normally month-day-year, though day-month-year will be allowed where recognizable. The separators are either -, /, space, or comma. Missing years will be defaulted to this year, and missing month to this month. If a day is not specified, then an entire month or year will be the input. This is one example of a Date-Time Range.

Month may be spelled out completely, trivially misspelled, abbreviated, or numbered. Days are numbered. Years are either four digits or two digits with 1900 assumed.

Last Tuesday and next Thursday are legal dates, as are today, tomorrow, and yesterday.

Times

The input format for times are hour:minute:second. This is normally expected to be 24-hour local time. Suffixes are recognized for other time zones, and AM & PM. Midnight and noon are also recognized. If the seconds or seconds and minutes are not specified, the time is a range covering the hour or minute specified.

Date-Time

A Date-Time is usually just a Date followed by a Time; however, now, 2 hours from now and 4 hours ago are legitimate Date-Times.

Date-Time Range

A Date-Time Range is most generally specified by a Date-Time followed by a later Date-Time. Several other range specifications have been given above. Additionally, tomorrow afternoon, next weekend, yesterday morning and this evening are also recognized.

Personnel Data Types

Rank/Name/Title/Organization/User-ID/Address

The personnel data types refer to users of the services. The users fall into three classes: individuals, positions, and organizations.

Special Categories

Message Type/Priority/Security Classification/Signoff

Special Handling

The definition of a special category data type is just a list of words and small integers associated with those words. If a particular argument to a function is listed as being a special category type, this means that the only legal inputs are words from a very restricted vocabulary, or an associated abbreviation. When the CLP recognizes the special category entry, the associated integer is passed to the Functional Module.

This is one data type class which the language designer can add to when he specifies the target environment. New special category data types, along with the Functional Module descriptions and user lexicon allow the laguage designer to mold the basic military message processing service to fit the needs of particular user groups.

Quantities

Cardinal/Dollar/Ordinal

The quantities are just numbers with specialized input and/or output formats. More data types will be added to this class as needed. As opposed to the special categories discussed above, these data types must be added by a CLP programmer. Some expected additions include telephone and Social Security numbers.

APPENDIX

Transmission Specification

Recipient/Receiver

The reviewer data type is only an output type. The information contained with a reviewer is a recipient (provided by author), signoff (provided by reviewer), and some text (provided by either). The input form for a recipient is different for a local or a remote addressee. For a local addressee the organization does not have to be specified. When the name or title is recognized as a local (this organization) one, and no organization is provided, the current one is assumed. If there is no recognition, the CLP will request the remote organization name or better spelling on the name or title.

Message Identifier

Message Identifier

A message is identified by three pieces of data related to it: the sponsoring organization, the creation or transmission date, and the creator or releasing authority (transmitter). When a user wishes to specify a message to the service, he must know the sponsor organization. After that a date-time or date-time range is necessary. The more specific the range, the easier the search. This is sufficient information for the CLP to aid the user in finding the message he is interested in. If the user does have the complete message identifier, that is of course helpful.

INTERNAL FORMATS AND OUTPUTS

This section lists all supported data types with their associate classes. Also enumerated are their internal formats and output forms, both short and long.

Address [Personnel Item]

Organization Number;
Name/Title Number.

For entire organization address, the user number is zero. For remote users, the user/title number is replaced by a text string containing the user/title name.

Short: Short Name/Title @ Short Organization

The local organization is omitted for short output form.

*>IPTO

*>Director

*>JCR @ IPTO

Long: Long Name/Title of Long Organization

The user's rank, where known, is used for the long output form.

*>ARPA/IPTO

*>Director Uncapher of USC/ISI

*>JCR Licklider, Director of ARPA/IPTO

Cardinal Number [Quantity]

Binary representation.

Short: Minimum decimal output

*>1 1234567890 -6

Long: 15 columns with leading sign and commas

*> +1 +1,234,567,890 -6

Character [Text Item]

Code number.

Short and Long: Character

Date [Chronological Item]

Number of days since Nov 17, 1858.

Short: Three character month followed by day number and two-digit year.

*>Jan 3,67

*>May 23,74

Long: Month followed by day number and year

*>January 3,1967

*>May 23,1974

APPENDIX

Date-Time [Chronological Item]

Number of days since Nov 17, 1858;
Number of seconds past midnight GMT.

Short: Short date followed by short time

*>June 17,75 17:52

*>December 23,56 9:13

Long: Long date followed by long time and time zone.
*>June 17,1975 5:52 PM PDT
*>December 23,1956 9:13 AM PST

Date-Times can be output in either local time or GMT.

Date-Time Range [Chronological Item]

Starting Date-Time;
Ending Date-Time.

Any Date-Time Range which corresponds to an entire year, month, day, hour, or minute will be output as the appropriate foreshortened Date-Time.

Short: Short start - short end

Long: Long start through long end

Dollar [Quantity]

Four bits per digit decimal character string representing number of cents.

Short: Minimal output with decimal point, though two trailing zeroes may be omitted.

Long: Dollar sign, commas, and trailing zeros are not omitted.

Line [Text Item]

Pointer to a block of characters terminated by a zero character;
Character count.

Short: The line is "filled" into the current display area.

Long: The line is "filled" and "justified" into the current display area.

Message Identifier [Message Identifier]

Organization Number;
Date-Time;
Name/Title;
Create or Transmit Flag;
File access information (on-line/off-line).

Short: Organization/Short Date-Time

*>CINCPAC/Jun 2,74-10:17:41

*>ARPA-IPTO/Sep 17,74-17:21:17

Long: Organization/Short Date-Time followed be either from (transmit) or by (create) and the user/title name.

*>CINCPAC/June 2,1974-10:17:41AM GMT by Major Wilcox

*>ARPA-IPTO/September 17,1974-5:21:17PM GMT from Kahn

Message Type [Special Category]

0 Informal
1 Formal

Short: FORMAL or INFORM

Long: FORMAL or INFORMAL

Name [Personnel Item]

user number or text string for remote user.

Short: user name

Long: Rank (if available) followed by user name

Ordinal Number [Quantity]

Binary number.

Short: Short cardinal followed by appropriate suffix (st, nd, rd, th).

*>1st,2nd,3rd,4th,5th,100th,101st

Long: Spelled out for numbers less than 100.

*>first,second,third,fourth,fifth,100th,101st

Organization [Personnel Item]

Organization Number.

Short: Organization abbreviation

Long: Organization Formal name

Priority [Special Category]

0	Routine
1	Priority
2	Immediate
3	Flash
4	Flash Override

Short: Priority Number

Long: Priority Name

Rank [Personnel Item]

Rank Number.

Short: Rank abbreviation

Long: Rank name

Recipient [Transmission Specification]

Organization Number;
Name/Title;
Priority Number;
Special Handling Code.

Short: Short address followed by priority number and special handling abbreviation

Long: Long address followed by priority and special handling

Reviewer [Transmission Specification]

Recipient;
Signoff Number;
Text.

Short: Short recipient followed by signoff abbreviation and text.

Long: Long recipient followed by signoff and text

Security Classification [Special Category]

0	Unclassified
1	Confidential
2	Secret
3	Top Secret

Short: Classification abbreviation

Long: Classification

Signoff [Special Category]

0	OK
1	OK?
2	OK-
3	Read
4	No Good
5	In Progress

Short: Signoff abbreviation

Long: Signoff

Special Handling [Special Category]

0	Eyes-only
1	No Forwarding

Short: Special handling abbreviation

Long: Special handling

Text [Text Item]

Pointer to characters terminated by a zero character;
Character count.

Short: "Filled" into current display window

Long: "Filled" and "justified" into current display window.

Time [Chronological Item]

Number of seconds since midnight.

Short: 24 hour time

*>10:17:21

*>17:21:45

Long: 12 hour AM-PM time with time zone

*>10:17:21AM EST

*>5:21:45PM GMT

Title [Personnel Item]

Title number.

Short: Title

Long: Title with name of current holder

Word [Text Item]

characters.

Short: Word

Long: Word

User-ID [Personnel Item]

Organization Number;
Name/Title Number.

Short: Name/Title @ Organization

Long: Name/Title of Organization